Software Energy Profiling: Comparing Releases of a Software Product

Jagroep, E.A.; van der Werf, J.M.E.M.; Procaccianti, G.; Lago, P.; Brinkkemper, Sjaak; Blom, L.; van Vliet, Rob

published in
Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume

2016

DOI (link to publisher)
10.1145/2889160.2889216

document version
Early version, also known as pre-print

Software Energy Profiling: Comparing Releases of a Software Product

Erik A. Jagroep,
Jan Martijn van der Werf,
Sjaak Brinkkemper
Utrecht University
Dept. of Information and Computing Sciences
Utrecht, The Netherlands
{e.a.jagroep, j.m.e.m.vanderwerf, s.brinkkemper}@uu.nl

Giuseppe Procaccianti,
Patricia Lago
VU University Amsterdam
Dept. of Computer Science
Amsterdam, The Netherlands
{g.procaccianti, p.lago}@vu.nl

Leen Blom, Rob van Vliet
Centric
Gouda, The Netherlands
{leen.blom, rob.van.vliet}@centric.eu

ABSTRACT

In the quest for energy efficiency of Information and Communication Technology, so far research has mostly focused on the role of hardware. However, as hardware technology becomes more sophisticated, the role of software becomes crucial. Recently, the impact of software on energy consumption has been acknowledged as significant by researchers in software engineering. In spite of that, measuring the energy consumption of software has proven to be a challenge, due to the large number of variables that need to be controlled to obtain reliable measurements. Due to cost and time constraints, many software product organizations are unable to effectively measure the energy consumption of software. This prevents them to be in control over the energy efficiency of their products.

In this paper, we propose a software energy profiling method to reliably compare the energy consumed by a software product across different releases, from the perspective of a software organization. Our method allows to attribute differences in energy consumption to changes in the software. We validate our profiling method through an empirical experiment on two consecutive releases of a commercial software product. We demonstrate how the method can be applied by organizations and provide an analysis of the software related changes in energy consumption. Our results show that, despite a lack of precise measurements, energy consumption differences between releases of a software product can be quantified down to the level of individual processes. Additionally, the results provide insights on how specific software changes might affect energy consumption.

ICSE '16 May 14 - 22, 2016, Austin, Texas, USA

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics

General Terms

Experimentation, Measurement, Performance

Keywords

Energy Efficiency, Profiling, Software Architecture, Software Product

1. INTRODUCTION

In the search for energy efficient solutions for the Information and Communication Technology (ICT) industry, research has mostly focused on hardware aspects in order to reduce the environmental impact of the sector. Indeed, every new generation of hardware improves its Energy Efficiency (EE) by either increased performance (i.e. more performance per Watt) or decreased Energy Consumption (EC) in absolute terms. Considering the growing number of hardware devices, the impact of these improvements can be significant. However, a crucial aspect that has been long overlooked is the role of software [17]. Although hardware ultimately consumes energy, software provides the instructions that guide the hardware behavior [28].

For example, the impact of software is clearly visible in the mobile phone domain. Although the EC of mobile applications is tipically closely monitored due to battery constraints [3, 19, 23], we have reached the point of requiring quad-core processors to ensure smooth operation. Nowadays, software updates require the user to buy a new mobile phone every few years, sometimes even without a clear benefit in terms of performance. Additionally, new phones are often equipped with higher capacity batteries, to prevent deterioration of the operation time.

Looking at larger software products, e.g. business applications, a similar pattern can be observed. Depending on the deployment, increasingly more powerful hardware is required to run new releases of applications. However, in contrast to the mobile domain, EC measurements with regard to business software products are more complicated to

perform. The diversity of deployments and levels of abstraction (e.g. virtualization and cloud computing) require more sophisticated measurement approaches to properly analyze software EC [25]. Recently, several of such approaches have been proposed, both hardware [6] and software based [22], which were able to identify opportunities for considerable savings in EC.

However, these approaches have not been adopted in industrial contexts so far. Although Software Product Organizations (SPOs), i.e. independent software vendors and open-source foundations, have software development as their core activity [12], having accurate software EC measurements still requires significant investments in terms of resources and specialized knowledge.

As a consequence, SPOs are unable to directly address software energy efficiency. For example, the Netherlands government specifies EC related requirements in their tenders. In practice, performance is often used as a proxy for energy efficiency. Software performance optimization is a more mature field of study, hence more people with such skills are available on the market. However, although much can also be derived from performance measurements, EC and performance are not always positively correlated; contradicting goals could require a trade-off to be made [11].

A deeper understanding of the matter is required to properly address the EC of the software itself. In this research we investigate a method that can be applied by SPOs to gain control over the EC of their software products. Given the dynamics of the software industry, such as multiple releases in relatively rapid succession, the method should explicitly address these dynamics by enabling an SPO to report any improvement or deterioration in EC with a new release. Our proposed profiling method is applied in an empirical experiment on a commercial software product.

The remainder of the paper is organized as follows: in Section 2 we present our research questions. In Section 3 we propose a profiling method to identify the EC of software. In Section 4, Section 5 and Section 6 we describe the design, execution and results of our empirical experiment. We discuss the threats to validity in Section 7 and present related work in Section 8. Concluding remarks and an outline for future work are provided in Section 9.

1.1 Contributions

The main contributions of this paper are:

Empirical study: To the best of our knowledge this is one of the few papers that compares the EC of software products across releases on a commercial software product.

Profiling method: We show in detail how to set up an environment to perform EC measurements across releases. The method includes the required tooling and metrics that enable an SPO to find the relevant energy hotspots [25] for their products and to quantify EC differences brought about by changes in the software.

Regression model: In addition to the method, we provide insight how to create a regression model to predict EC more accurately based on performance and EC data. With this regression model we add to the 'green mining' research area [9].

2. RESEARCH QUESTIONS

Based on the problem explained in Section 1 we formulate our **main research question** as follows:

RQ: *How can we reliably compare the energy consumption of large scale software products across different releases?*

In the RQ, we explicitly refer to large-scale software products as multi-tenant, multi-user distributed software applications, as opposed to e.g. single-user mobile applications which are out of scope for our research.

A prerequisite for comparing the EC of a software product is being able to measure the software EC. Therefore our first research sub-question is:

SQ1: *How can we reliably measure the EC of a software product?*

Software-only approaches can be roughly categorized in two sets: (source code) instrumentation [22] and energy profilers [10]. Hardware-based approaches (e.g. [5]) rely instead on physical power meters to be connected to hardware devices. Given our focus on large scale software products, we see two potential issues with source code instrumentation: the *complexity* overhead and the *investment* that is required to apply them in terms of software development skills. Hence, we do not see them as usable in an industrial setting. Software energy profilers do not require a high effort to be adopted, but are shown to be inaccurate in their measurements [10]. On the other hand, hardware-based approaches do not provide fine-grained measurements at software level, i.e. they are not able to trace the energy consumption of single software elements such as processes or architectural components. In Section 3 we propose a profiling method that leverages both measurements: we use software profilers to obtain fine-grained, software-level estimations and we validate them with hardware measurements obtained via power meters.

Our proposed profiling method is applied in an empirical experiment on a commercial software product, described in Section 4 and Section 5. For SPOs to actually be able to influence the EC, changes in EC should be related to the individual software elements. To this end we formulated two more sub-research questions (Section 6):

SQ2: *How can we attribute energy consumption to individual software elements?*

SQ3: *How can we relate differences in energy consumption to changes in a software element?*

The second sub-question (SQ2) is set to investigate how the EC is divided over the combination of software elements that comprise the software product. Since software measurements can provide these details, we include in the evaluation a software tool, Joulemeter, to fulfill this purpose. After relating EC to software elements, following SQ3, we analyze the impact of software changes on EC.

3. PROFILING METHOD

To better understand our experiment design, this section describes the method we adopted to profile the EC of a software product. As a basis for the measurement method, we use the work presented in [8, 11, 14].

3.1 Hardware– and Software–Based Measurements

A measurement method concerning software EC should include both hardware and software approaches to obtain the right level of detail in the measurements. In terms of

hardware measurements, we rely on power metering devices. As these meters are installed between a device and its power source, a meter is required for each power supply unit of the devices under test. Although these meters are capable of achieving high levels of accuracy, their specifications should be taken into account in the analysis as even small measurement errors might be significant when measuring at software level.

Regarding software profilers, we require such tools to not only estimate the total energy consumption of the system, but also to profile individual software processes. Unfortunately, although a more fine-grained interval is desired [9], these tools record measurements with a one second interval. While the usability and accuracy of energy profilers still have margins for improvement [10, 20], the reported measurements could still be used to detect differences in EC. In other words, although measurements in absolute terms may not be fully accurate, the relative differences between EC of releases can still provide useful insight.

3.2 Performance Measurements

In addition to the EC, the hardware performance needs to be recorded. Inspired by the metrics provided in [2, 11, 14], performance data could fill the gap when it comes to accurately relating EC to individual software elements. To do so, performance data must be collected at both hardware and process level.

The configuration of performance profiling software requires the user to have a basic understanding of the hardware components that have to be monitored through so-called "performance counters". In addition, when interpreting performance data for further analysis, context information has to be taken into account (e.g. hardware-specific details). Following the definition of the 'Unit Energy Consumption' [11], in our experiment we set up performance counters for the most frequently monitored hardware resources:

- Hard disk: disk bytes/sec, disk read bytes/sec, disk write bytes/sec
- Processor: % processor usage
- Memory: private bytes, working set, private working set
- Network: bytes total/sec, bytes sent/sec, bytes received/sec
- IO: IO data (bytes/sec), IO read (bytes/sec), IO write (bytes/sec)

3.3 Idle EC, Software Overhead and Cooldown Time

The aim of the method is to extract the EC of the software under test. Therefore, part of the method is to identify the amount of EC for which the software is responsible. Calculating the *Software Energy Consumption* (SEC) [11], requires the idle EC for the hardware that is used. This is then subtracted from the total EC during a measurement, assuming the increase in EC solely depends on running the software under test. As the idle EC heavily depends on the used hardware, this number should be determined separately for each hardware device in the experiment. Determining the idle EC simply means to perform measurements while the hardware is running without any active software. As measurement software causes EC as well, it should be part of the idle measurement, not to pollute any measurement.

Ideally, software measurements should be performed remotely to minimize the overhead on EC, caused by the data collection process. Notice that the SEC also includes Operating System (OS)-specific activities (e.g. background daemons), which we are not (yet) able to consider separately and thus considered to be part of the idle measurement.

Another software related aspect is the cooldown time a server needs after rebooting. After a reboot, several services related to the OS are active without direct instructions from a user. As these services require computational resources, they most likely will pollute measurements if the experiment starts while these services are executing. When the extra services become inactive, there is a correct basis to start measuring. To determine the cooldown time, which should be determined for every hardware device included in the experiment, EC and performance measurements should be analyzed to determine after what time the measurements become stable.

3.4 Data Synchronization

An important requirement for data analysis is to have synchronized measurements. As measurements are obtained from different sources, their time system should be synchronized, since otherwise the measurements become incompatible. For example, if a specific activity is performed and the time data across sources is not in sync, there is a risk of missing the data related to this activity. A simple solution to this problem is to synchronize the clocks for all measurement instances using the Network Time Protocol (NTP).

3.5 Measurement Protocol

An important aspect in the experiment design is to have a protocol to perform measurements. This improves reliability of each measurement and ensures consistency across measurements [30]. In [8], the "green mining" method is presented to measure and extract power consumption data relevant to software change consisting of seven activities that should be performed: (1) choosing a product and context, (2) decide on measurement and instrumentation, (3) choose a set of versions, (4) developing a test case, (5) configure the testbed (6) of each version and configuration, and (7) compile and analyze the results. Compared with this method, a measurement protocol should be considered as part of the 6th activity where a measurement is performed for each version and configuration included in the experiment. The activity is further specified into (a) run the test within the testbed and record the instrumented data, (b) compile and store the recorded data and (c) clean up the test and the testbed.

While the "green mining" method of [8] provides a solid basis for designing an experiment, serving its purpose as a generic method, no details are provided on how to actually perform valid, reliable measurements within an experiment. This can be a barrier for its adoption in practice, as an SPO needs practical guidelines and more details with each activity in the protocol. To this end, we propose the following measurement protocol, which is an extension to the activities presented by [8]:

i Restart environment; (c)

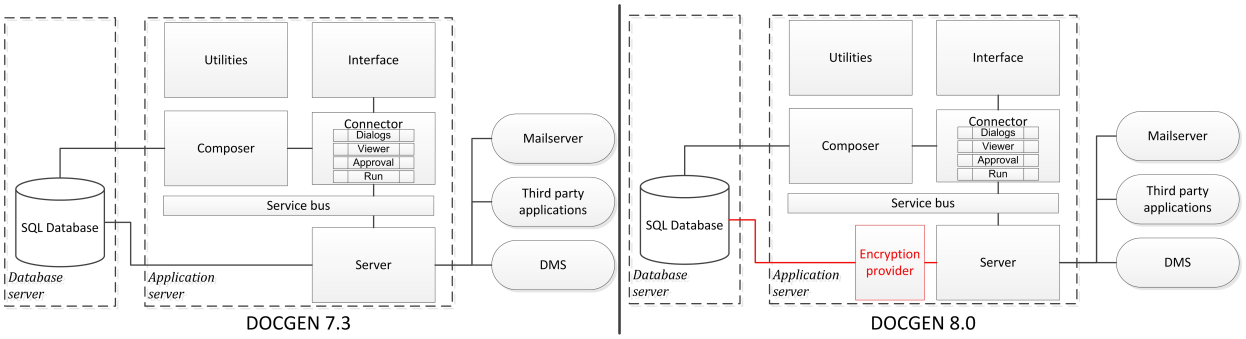

Figure 1: The functional architectures for Document Generator (DG) releases 7.3 (left) and 8.0 (right) portrayed on a commercial deployment. The changes are in red.

ii Check time synchronization;

iii Close unnecessary applications;

iv Start performance measurements; (a)

v Remain idle for a sufficient amount of time;

vi Start EC measurements; (a)

vii Run measurement and wait for run to finish; (a)

viii Collect and check data; (b)

ix Revert environment to initial state; (c)

Notice that a large part of the protocol can be mapped to parts of the existing method (letters between brackets).

3.6 Metrics

Comparing the "green mining" method with the methods applied in [11] and [14] we find similarities in the measurement method that is applied, but a clear difference in the reported metrics. Although both report EC, the reported metrics target different stakeholders while still providing the details required to be in control of the software EC. During the design of an experiment, a choice should be made on what metrics are to be reported, as they should facilitate discussion between stakeholders, e.g. product managers and (potential) customers [4], especially in the case of a pioneering topic like the EC of software [6].

4. EXPERIMENT DESIGN

To demonstrate the ability of the proposed method to compare EC of a software product across different releases and to explain differences at software architecture level, an experiment is performed on a commercial product, Document Generator (DG). For the experiment we applied the method described in Section 3 and followed the guidelines provided in [13, 16, 26, 30].

4.1 Product Under Study: DG

Document Generator (DG) is a commercial software product that is used to generate a variety of documents ranging from simple, informative mailings to complex documents concerning financial decisions. The product is used by over 300 organizations in the Netherlands, counting more than 900 end-users, and annually generates more than 30 million documents. This experiment focused on two releases of DG, 7.3 and 8.0, allowing us to compare the effects of a major release [31].

In Figure 1 the Software Architecture (SA) is shown for the DG releases included in the experiment. Starting with the *Connector* element, we have a central hub in the SA responsible for receiving user input through the *Interface*, collecting data from the *Composer* and handling communication with the *Service bus*. Together with the *Composer* element, responsible for merging document templates and definitions with database data, the *Connector* element handles all activities before documents are generated. *Utilities* and *Interface* respectively provide configuration options and an interface for DG. The final element on the application server is the *Server* element responsible for the actual generation of the documents and delivering the documents to where they are required. The database server hosts the Oracle *SQL Database*.

4.1.1 Differences between Releases

Looking at the SA, the major difference is the encryption provider introduced in release 8.0 on the application server. Data encryption was introduced in release 8.0 in order for DG to comply with the upcoming General Data Protection Regulation (GDPR) set up for the European Union. In the case of DG 'Microsoft Enhanced Cryptographic Provider' is used; a module that software developers can dynamically link to when cryptographic support is required. Encryption is applied in relation to the 'Server' element to remain independent from the database that is used; i.e. encrypted data is sent to the database

Another difference, not visible in the SA, can be found in the data model for the database. As release 8.0 is compliant with a new document management system, the datastructure is more complicated compared to release 7.3. Cross-checking our findings with the DG architect ensured completeness of our list of relevant changes for the experiment.

4.1.2 Test Case

For the experiment we chose to stress DG with its core functionality, being the generation of documents. DG was instructed to erase existing documents of a certain type and consecutively regenerate these documents. The document type that was used contains both textual information and financial calculations and a total number of 5014 documents was generated per measurement. During a measurement

Table 1: Specifications of the hardware and software used for the experiment.

	Application server	Database server
Hardware	HP Proliant G5, 2 x Intel Xeon E5335 (8 cores @ 2GHz), 8 GB DDR2 memory, 300 GB hard disk @ 15.000 RPM	HP Proliant G5, 1 x Intel Xeon E5335 (4 cores @ 2GHz), 8 GB DDR2 memory, 300 GB hard disk @ 15.000 RPM
Operating system	Windows 2008 R2 Standard (64-bit), Service Pack 1	Windows 2008 R2 Standard (64-bit), Service Pack 1
Software	DOCGEN 7.3 and 8.0	Oracle 11.0.2.0.4.0

the 8 processes 'Interface', 'Run', 'Connector', 'Server', 'Oracle', 'TNSLSNR', 'omtsreco' and 'oravssw' processes are to be monitored on their respective servers. As the 'Microsoft Enhanced Cryptographic Provider' is not an executable but a dynamic library, we were not able to perform measurements on this element.

4.2 Experiment Environment

In line with the deployment portrayed in Figure 1, two servers have been used: one for the application and one for the database. Each of the servers has its own WattsUp? Pro (WUP) device. The setup is depicted in Figure 2. The specifications of the application and database servers are provided in Table 1.

Both releases of DG were installed on the application server and Oracle was installed on the database server. The setup of the experiment, including the servers are comparable with a commercial setting of the product. In the experiment, both releases use the same data set of an actual customer.

The cooldown time for these server was determined to be 15 minutes. A separate logging server is added to the setup to collect experiment data to keep the EC measurements as clean as possible. To ensure consistency with regard to external factors (e.g. room temperature), the servers were installed in an operational data center.

4.3 EC and Performance Measurements

For the experiment we want to measure the SEC and UEC metrics as defined in [11], at the level of the concurrent processes of the product.

To mearue the EC, we use WUP devices which record the total energy consumption of the hardware per second. As these measurements are at server level, further processing is required to obtain measurements related to the software.

On the software side EC data is collected using the tool Joulemeter (JM) of Microsoft, that allows to estimate the power consumption of a system down to the process level. JM estimates EC on a model that first needs to be calibrated for the hardware it runs on. Previous experience

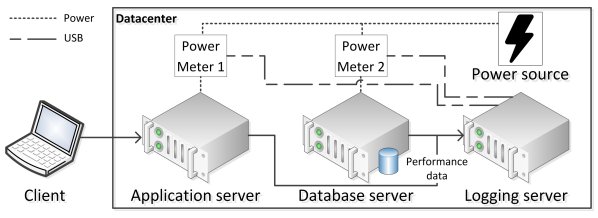

Figure 2: Experiment environment.

with JM [10] shows that altough JM provides a general idea of EC, it differs significantly from the actual EC. Since only one process can be measured per instance of JM, a separate instance for each of the concurrent DG processes is instantiated (see Section 4.1).

The performance of the application and database servers are measured using the standard performance monitor (perfmon) provided with Microsoft Windows. As perfmon does not include network performance counters at process level, we exclude these from the experiment. All other performance counters (Section 3.2) are included. Performance data is remotely collected using the logging server, thereby minimizing the overhead of measurement on the actual hardware.

4.4 Protocol

To ensure consistency across measurements, we follow the protocol defined in Section 3.5. To increase the consistency across measurements, a script is used to generate 5014 documents using DG.

Summarizing the data collected for each individual measurement we have:

- WUP measurements of the energy consumption at the level of the hardware;
- JM estimates for each of the processes together with an estimate of the total energy consumption;
- one perfmon file containing the performance measures for both the application and the database server;
- the start and end timestamp for each measurement;

After each measurement, both servers have been reverted to the initial state. To mitigate the risk of mismatched time data, all devices are continuously synchronized using the Network Time Protocol (NTP).

5. DATA COLLECTION

In this section we report on our data collection process. Both the WUP as well as the JM measurements report the EC as an average of the instantaneous power over the sampling interval. To calculate the total EC, we either multiply the average power with the time the system was running, or sum up the recorded energy measurements. We report our findings in Watt (W) or Watthour (Wh) where applicable.

5.1 Server EC Characteristics

The results of the idle and JM overhead measurements are presented in Table 2 along with the measurement time to determine the averages. Starting with the idle EC we found an average power consumption of 274.54 W for the the

Table 2: Comparison of server power consumption in the idle and idle with JM scenarios.

Server	Idle		Idle with JM		Idle with JM according to JM	
	Total time	Avg. Power (W)	Total time	Avg. Power (W)	Total time	Avg. Power (W)
Application	57:11:30	274.54	54:06:21	275.28	54:06:21	276.18
Database	57:11:30	252.59	54:06:21	252.79	54:06:21	253.39

application server and of 252.59 W for the database server. Considering that the servers are almost identical, we can only allocate this difference of 21.95 Watt (W) to the extra processor available in the application server.

An interesting finding concerning JM is the fact that there is minimal to no overhead on the account of this software. Further investigation showed a base memory usage by JM, which increased when JM was actually logging measurement data. While logging, performance measurements show increases in the memory usage of the JM instances which are periodically 'reset' to a base memory usage. Our guess is that the pattern in memory usage corresponds to incrementally adding measurements to the CSV file. Despite this variability in memory usage we could not detect any change in EC. If we use the JM measurements to determine the average power consumption (right of Table 2), a larger difference is perceived which is in line with the findings presented in [10].

5.2 DG measurements

We performed 20 measurements for each DG release (7.3 and 8.0) where the thirteen items (described in Section 4) were collected per measurement. Table 3 summarizes the results of these measurements in terms of the averages for respectively the application and database server. Notice that the measurements for the database server only show the JM results for the 'Oracle' process. The other processes were excluded from the table as we found no EC on their behalf during a measurement, despite them being active. The same holds for the 'Interface' process on the application server, which, by using a script, was not activated during the experiment.

Comparing the measurements between releases, two differences are clearly visible. First is the difference in average run length of twelve seconds, which is surprising considering the fact that the scripts used to stress both releases were identical. A second difference is the overall increase in energy consumption of DG 8.0 as compared to 7.3 with 4.14 Wh according to the WUP measurements; 2.97 Wh for the application server and 1.17 Wh for the database server. An increase that, to a lower extent, is also reflected in the JM data. If the 'idle with JM' EC is subtracted from these differences for twelve extra seconds, to remove the effect of having longer measurements, we still find a difference of 2.05 Wh and 0.32 Wh that is on the account of DG.

The SEC for both DG releases is calculated by subtracting the 'idle with JM' EC from the total EC as reported by the WUP for the length of the run. These EC figures are obtained by calculating the area under the power consumption curve. For release 7.3 we find a SEC of 2.57 Wh for the application server and 8.03 Wh for the database server. Measurements for release 8.0 provide a SEC of 4.61 Wh and 8.34 Wh for the application and database server.

5.3 Joulemeter Estimations

The SEC can also be calculated using the EC estimations provided by JM. Using this data we find a SEC of 1.45 Wh for the application server and 5.69 Wh for the database server with release 7.3 and 1.57 Wh and 5.72 Wh with release 8.0. Straightaway we notice the differences between these SEC figures and the ones obtained using WUP. In our data we observe that the WUP on average provides a higher SEC of 1.22 Wh and 2.34 Wh for the application and database servers, a difference that can only be caused by the JM power model.

Apart from the total EC, the JM data can also be used to calculate the SEC according to the measurements on process level, i.e. the 'Run', 'Server' and 'Connector' processes at the application server and the 'Oracle' process on the database server. The measurements for release 7.3 provide a SEC of 0.89 Wh and 5.69 Wh for the application and database server. With release 8.0 we find a SEC of 0.97 Wh and 5.62 Wh respectively. The large differences in the SEC figures could be an indication of the multitude of processes that become active in the background alongside the DG processes.

6. EXPERIMENT RESULTS

In this section we further analyze the results obtained in our experiment and answer our research sub-questions.

6.1 SQ1: Measuring the EC

The profiling method that was applied in the experiment, encompasses activities to ensure that the relevant variables (that can be influenced) are under the control of the researcher. It also provides guidelines for the data collection and processing. By following the measurement protocol we obtained consistent and comparable data across measurements, confirmed by the small standard deviations found with each item. We also did not come across any peculiarities while collecting and processing the data, leading us to the finding that, **using the measurement method as described, allows us to reliably measure the EC of a software product**.

6.2 SQ2: Relating EC to Software Elements

The percentages of EC that JM leaves unexplained on process level (on average 61.9% at the application server and 69.3% on the database server) indicate that we are still unable to explain a relatively large amount of the energy overhead of software execution.

One possible explanation is a lack of accuracy of JM. The profiling tool is based upon a linear model that takes into account only a limited amount of hardware resources [15]. Hence, it is reasonable to conclude that this energy estimation gap is due to unaccounted resources in the linear model. For this reason, we tried to build a special–purpose linear model, trained by using performance data and the energy consumption measured by the WUP. The model was built by means of penalized linear regression [29], a regression technique that enables to specify constraints for the model

Table 3: Summary of the measurements performed on the servers for both DG releases.

		Application server				Database server			
		7.3		8.0		7.3		8.0	
		μ	σ	μ	σ	μ	σ	μ	σ
Run length (hh:mm:ss)		2:48:16	4 s	2:48:28	7 s	2:48:16	4 s	2:48:28	7 s
Processed Documents		5014		5014		5014		5014	
WUP (Wh)		774.59	1.18	777.56	0.84	716.99	0.45	718.16	0.61
Run	Total (Wh)	765.20	0.32	766.21	0.63				
	Process (Wh)	0.0002	0.00009	0.0003	0.0001				
Server	Total (Wh)	765.18	0.33	766.21	0.63				
	Process (Wh)	0.744	0.00002	0.758	0.007				
Connector	Total (Wh)	765.19	0.34	766.22	0.63				
	Process (Wh)	0.144	0.004	0.22	0.004				
Oracle	Total (Wh)					706.37	0.29	707.27	0.51
	Process (Wh)					5.63	0.02	5.62	0.02

features. This was done in order to enforce a positive value for the predictors. This assumption builds upon the rationale that a software process will use a positive and finite share of the system resources.

Our special-purpose model outperforms JM at machine-level prediction i.e. trying to predict the total system EC, see Figure 3. The model has a MAPE of 0.004 when compared to WUP measurements, whereas JM has 0.005. However, the process-level prediction is quite overestimated, probably due to an incorrect determination of the intercept term. This is a strong indication that other factors are playing a role. Examples might be networking devices, or OS-level processes and system calls that the profiler is unable to detect as separate processes. Hence, further work must be done to reliably attribute EC to specific software elements.

That being said, **our profiling method allows us to observe relevant changes between the different processes composing our software product** DG. This allows us to make informed hypotheses about the impact of each elements on our software product. For example, the 'Oracle' process in the database server is by far the most energy–consuming. This indicates that the database is a potential *energy hotspot* [25] and, as such, a candidate for optimization.

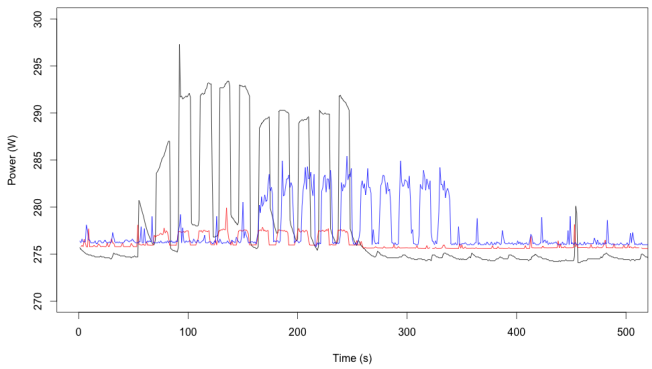

Figure 3: Performance of our special-purpose regression model (in red) vs. Joulemeter (in blue). Measured values by WUP are in black.

6.3 SQ3: Relate EC Differences to Software Changes

The most apparent difference between the DG releases is the introduction of the encryption provider element on the application server. Unfortunately, as this element is a dll, we were not able to perform measurements specifically on this element and thus could not be included in the SEC figures. We are, however, able to analyze the effects that are caused by the addition of this elements and infer possible explanations for EC differences.

According to the architect, the introduction of the encryption provider was accompanied by minor changes in the 'Server' element. Interestingly though, while an increase in EC is found in the 'Server' element, the main EC difference was found in the 'Connector' element going from 0.144 Wh to 0.215 Wh. A difference that could not be explained based on the adjustments applied in release 8.0. This *unforeseen change* in EC was reason for the architect to further investigate the matter in the near future.

With regard to the difference in run length an explanation is sought in the encryption that is applied, possibly extending the time required to set up a connection and communicate data. Apart from increased duration of the run, we also found that the net number of seconds that JM reports that energy is consumed increases with release 8.0 for the 'Server', 'Connector' and 'Oracle' processes. Combining this finding with the linear model applied by JM, more seconds of measurement, i.e. more activity, should mean a higher EC for these processes. However, this only holds for the processes running on the application server.

Overall we can conclude that the changes applied in release 8.0 increased the SEC with 4.14 Wh for the generation of 5014 documents. Presenting the results to the stakeholders of DG, they are now able to not only quantify, but also justify changes in EC. Considering the cause of this increase, being compliant with a new document management system and ready for the General Data Protection Regulation, the stakeholders accept the increase in EC. To increase efficiency, however, the software architect will still look into the 'Connector' element.

The results in this section show that, **by using the results of the profiling method, we are able to think of grounded explanations for differences found in the EC across releases of a software product**. Although our profiling method requires more detailed measurements

to draw hard conclusions, we are able to provide guidance when EC aspects are discussed and point out possible unexpected differences.

7. THREATS TO VALIDITY

This sections presents the threats to internal, external and construct validity as required by [13, 26, 30].

7.1 Internal Validity

The internal validity is concerned with the uncontrolled factors that might affect the results of the experiment.

JM reliability. Although we were able to clearly identify differences between the estimated energy consumption of the selected processes, the estimations only accounted for percentages of the variation in EC. A brief cross–validation, conducted by means of a self-obtained regression model based on resource consumption information, reveals a much higher impact of single processes on total energy consumption than estimated by Joulemeter. Hence, additional work is needed to have a clear and reliable attribution of the energy impact of single processes.

Measurement Interval. Both hardware and software measurement approaches have a sampling interval of one second. Given the nature of electrical power, this low sampling frenquecy might result in an underestimation of energy consumption due to high-frequency energy components. However, this interval is also commonly applied in the state of the art [8].

OS Effects. In the experiment the EC of the OS was included in the reported SEC for DG as we could not measure the OS separately during a measurement. Ideally, the OS would be considered as a separate layer with its own, distinguishable EC. Also, there is the possibility of OS processes and services that might become active during a measurement without a direct, controllable trigger. Deep analysis of the performance measurements could show whether such an activity has occurred during a measurement, provided that the activity can be measured to begin with.

EC Overhead. The EC of software not related to DG was measured and taken into account (as overhead) while calculating the SEC. These measurements were performed separately to obtain clean overhead figures. However, by doing so we do not include any effect of having multiple software applications running simultaneously. Further research is required to fully understand and control this effect.

7.2 External validity

The external validity addresses the extent to which the results can be generalized beyond the experiment.

Experiment Setting. Our experiment is limited to a single application and tested on a single testbed. Hence, we cannot generalize the effect size of changes in the EC on our target population of commercial software products. Nevertheless, we argue that our work can be useful to generate awareness in software developers and architects about the knowledge gap in software energy efficiency.

Hardware Specificity. One of the main factors that could influence the EC measurements is the specific hardware; new generations of hardware often boast improved performance and EE. For this reason we explicitly added, among others, idle EC in the measurement method, to create a matching EC profile for the hardware. We argue that differences might be found when comparing the absolute numbers, but that the relative proportions should be consistent across different hardware setups.

Measurement tooling. Both hardware and software measurement approaches are applied to obtain the experiment data. Given the diversity of power meters and software tools available, each with their own advantages and limits, there is an unavoidable dependency on the equipment when it comes to the accuracy and detail of the measurements.

7.3 Construct validity

Construct validity addresses the degree to which the measures capture the concepts of interest in the experiment.

Metrics vs. outcome. A central aspect in performing EC measurements is to have a clear view on the metrics that should be reported. In the experiment method we included a section on choosing the appropriate metrics for the experiment and the stakeholders. In our experiment design, the measurements reflect the data required to calculate the metrics. With regard to the metrics themselves, a solid list is already available in the literature [2].

Definition of change. Our goal is to relate software changes with their effects on the EC. Although we can empirically assess the difference between the energy consumption of the two application releases, we do not aim to provide a general definition of what a 'change' represents in software. For that purpose, we simply use two different releases of the DG product. Then, we provide insight as to which specific changes could affect the observed difference in EC. Further work is needed to pinpoint (and predict) the exact energy consumption impact of a generic software change.

8. RELATED WORK

Measurement method: EC measurements on mobile devices are commonly performed to prevent the software from having a deteriorating effect on the battery life of the device., e.g. by software tools performing measurements on the device itself (Joulemeter [7], eprof [23]), or by emulation tools that allow developers to estimate the EC of their application on their development stations [19]. Since battery drain can be monitored relatively easily and mobile devices have similar hardware architectures, these tools provide accurate results.Additionally, as performance profilers are quite mature in mobile computing, EC profilers can build upon such tools [18].

EC profiling is less common in the area of large scale software products, where we can identify multiple approaches to enable measurements in complex environments. A commonly accepted approach is to use performance measurements to explain and characterize software and its EC characteristics [2, 14]. Others focus on the power models [20] used by multiple tools and recommend the power models to become more fine-grained to deliver more accurate measurements at software level. Finding a regression model using EC and performance data could be considered part of green mining [9]. Unfortunately, due to lack of publicly available data, green mining is still an immature area.

A different approach is to supply a software solution promoting the energy efficiency of the software. The 'Eco' program [33], for example, introduces energy and temperature awareness in relation to the software and involves developers in the loop of finding energy friendly solutions. Although different information is used, the 'JalenUnit' [21] can be used to, a.o., detect energy bugs and understand energy distribu-

tion. The 'JalenUnit' infers the energy consumption model of software libraries from execution traces. Despite the differences in approach and accuracy of the results, measurement methods are all on the lookout for energy hotspots [22].

EC comparison between releases: Comparing aspects across releases is often discussed in terms of software evolution [27]. However, only few papers were found that investigate the EC of software and include a comparison between different releases. In [8] a comparison is made between three releases of rTorrent by 'mining'EC and performance data. A direct relation is described between the granularity of the measurements and the ability to determine the cause of changes in EC. Another approach is to characterize software using Petri nets [32]. Assumed that a complex software product can be fitted into a Petri net, analysis could show the path of lowest EC to perform a specific task. If the changes in a new release can be included in the Petri net, the difference(s) between releases can be quantified.

Awareness: Awareness of the software community about the impact of software on EC is increasing [1]. However, Pinto et al. [24] point out that this is still far too little to make a difference. In spite of recent progress, also the state of the art of EE software did not yet reach sufficient quality to deliver reliable detailed measurements. Comparing the EC between releases can be used to start creating awareness at the right place for a SPO, and hence exert control over their software in terms of EC.

9. CONCLUSIONS

In this paper we present the results of an experiment performed on the EC of a commercial software product. We consider the perspective of a software product organization aiming to exert control over the EC of its software products, and posed the following **main research question**: 'How can we reliably compare the energy consumption of large scale software products across different releases?'. We provide an answer to this question by investigating three sub-research questions.

To reliably measure the EC of a software product (SQ1), we constructed a profiling method. The topics discussed in the method provide hands-on instructions to create an energy profile of the hardware, obtain EC data that is consistent across measurements and ensure that the data is ready for analysis. As an initial validation, our profiling method was successfully applied in an industrial setting, on a commercial software product. The reported metrics proved useful to communicate and discuss the results with industrial stakeholders.

The second and third sub-questions aim at providing an SPO with the required information to actually control the EC of a software product. Starting with relating EC to individual software elements (SQ2), our profiling method includes the estimation of EC at process level by means of energy profilers. Our analysis showed that energy profilers can only explain percentages of the total EC for the application server and an even lower percentage for the database server. We tried to find a regression model to fill this gap in the data, but were (yet) unable to create an accurate model at the process level. However, our method successfully identified changes in EC at process level.

The final sub-question (SQ3) addresses how changes in EC can be related to changes in the software elements. This requires a clear overview of the changes that are made and an energy profile of the application. Any differences found in the measurements between releases are considered to be caused by at least one of these changes and as such should be further investigated using the data available. Ideally, aspects of the energy profile can be related to the individual software elements in order to find quantifiable possible explanations for any changes in the EC. Our experiment showed that the total EC of DG increased with release 8.0 w.r.t. 7.3. While this increase was expected, actual EC data (provided by the WUP) now verifies it quantitatively. Relating the increased EC to the changes in the software, stakeholders deemed this increase as justifiable, and the SPO experts could use the quantification to establish a better causation link.

Our proposed method leverages the possibilities of software energy profiling. An SPO that applies it can compare the EC of their products across different releases and gather insights in where adjustments should be made to decrease EC. In future work, we plan to apply our method during the development phase, aiming to provide software developers with direct EC feedback while developing. As we acknowledge that other views and analysis on our data could yield interesting findings, the complete dataset of the experiment is openly available [1]. We strongly encourage other researchers to contribute to this field of research.

10. ACKNOWLEDGMENTS

We would like to thank Edwig Huisman, Yuri Idris and Ronald Roos for their help in setting up the experiment and actively proposing and discussing possibilities to improve the experiment, and Fabiano Dalpiaz, Garm Lucassen and Leo Pruijt for their valuable discussions and feedback.

11. REFERENCES

[1] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. Venters. Sustainability Design and Software: The Karlskrona Manifesto. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 467–476. IEEE, May 2015.

[2] P. Bozzelli, Q. Gu, and P. Lago. A systematic literature review on green software metrics. Technical report, Technical Report: VU University Amsterdam, 2013.

[3] H. Chen, B. Luo, and W. Shi. Anole: A case for energy-aware mobile application design. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 232–238, 2012.

[4] C. Ebert and S. Brinkkemper. Software product management - an industry evaluation. *Journal of Systems and Software*, 95(0):10 – 18, 2014.

[5] M. A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, and J. Visser. Seflab: A lab for measuring software energy footprints. In *GREENS*, pages 30–37. IEEE, May 2013.

[6] K. Grosskop and J. Visser. Identification of application-level energy optimizations. *Proceeding of ICT for Sustainability (ICT4S)*, pages 101–107, 2013.

[7] A. Gupta, T. Zimmermann, C. Bird, N. Nagappan, T. Bhat, and S. Emran. Detecting energy patterns in

[1]Dataset available via: https://www.dropbox.com/sh/kk9kastzo2cypur/AABA3ZuWbSi-F4k8o8Af6KJJa?dl=0

software development. *Microsoft Research Microsoft Corporation One Microsoft Way Redmond, WA*, 98052, 2011.

[8] A. Hindle. Green mining: a methodology of relating software change and configuration to power consumption. *Empirical Software Engineering*, pages 1–36, 2013.

[9] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 12–21, New York, NY, USA, 2014. ACM.

[10] E. Jagroep, J. M. E. M. van der Werf, S. Jansen, M. Ferreira, and J. Visser. Profiling energy profilers. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 2198–2203. ACM, 2015.

[11] E. A. Jagroep, J. M. E. M. van der Werf, R. Spauwen, L. Blom, R. van Vliet, and S. Brinkkemper. An energy consumption perspective on software architecture. In *Working IEEE/IFIP Conference on Software Architecture*, number 9278 in LNCS, pages 239–247. Springer, 2015.

[12] S. Jansen, S. Brinkkemper, J. Souer, and L. Luinenburg. Shades of gray: Opening up a software producing organization with the open software enterprise model. *Journal of Systems and Software*, 85(7):1495–1510, 2012.

[13] N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[14] G. Kalaitzoglou, M. Bruntink, and J. Visser. A practical model for evaluating the energy efficiency of software applications. In *ICT for Sust. 2014 (ICT4S-14)*. Atlantis Press, 2014.

[15] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 39–50, New York, NY, USA, 2010. ACM.

[16] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002.

[17] P. Lago, R. Kazman, N. Meyer, M. Morisio, H. A. Müller, F. Paulisch, G. Scanniello, B. Penzenstadler, and O. Zimmermann. Exploring initial challenges for green software engineering: summary of the first GREENS workshop, at ICSE 2012. *ACM SIGSOFT Software Engineering Notes*, 38(1):31–33, 2013.

[18] Y. Liu, C. Xu, and S.-C. Cheung. Characterizing and detecting performance bugs for smartphone applications. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1013–1024. ACM, 2014.

[19] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 317–328, New York, NY, USA, 2012. ACM.

[20] A. Noureddine, R. Rouvoy, and L. Seinturier. A review of energy measurement approaches. *SIGOPS Operating Systems Review*, 47(3):42–49, Nov. 2013.

[21] A. Noureddine, R. Rouvoy, and L. Seinturier. Unit testing of energy consumption of software libraries. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1200–1205, New York, NY, USA, 2014. ACM.

[22] A. Noureddine, R. Rouvoy, and L. Seinturier. Monitoring energy hotspots in software. *Automated Software Engineering*, pages 1–42, 2015.

[23] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conf. on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA, 2012. ACM.

[24] G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 22–31, New York, NY, USA, 2014. ACM.

[25] G. Procaccianti, P. Lago, A. Vetro, D. M. Fernández, and R. Wieringa. The green lab: Experimentation in software energy efficiency. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 2015.

[26] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

[27] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. Nasser, and P. Flora. An exploratory study of the evolution of communicated information about the execution of large software systems. *Journal of Software: Evolution and Process*, 26(1):3–26, 2014.

[28] Y. Sun, Y. Zhao, Y. Song, Y. Yang, H. Fang, H. Zang, Y. Li, and Y. Gao. Green challenges to system software in data centers. *Frontiers of Comp. Sc. in China*, 5(3):353–368, 2011.

[29] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Series B Stat. Methodol.*, 58(1):267–288, 1 Jan. 1996.

[30] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[31] L. Xu and S. Brinkkemper. Concepts of product software. *European Journal of Information Systems*, 16(5):531–541, 2007.

[32] G. Zhang, K. Zhang, X. Zhu, M. Chen, C. Xu, and Y. Shao. Modeling and analyzing method for cps software architecture energy consumption. *Journal of Software*, 8(11), 2013.

[33] H. Zhu, C. Lin, and Y. Liu. A programming model for sustainable software. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 767–777, May 2015.